Energy Optimization by Exploiting Execution Slacks in Streaming Applications on Multiprocessor Systems

Amit Kumar Singh, Anup Das, Akash Kumar
Department of Electrical and Computer Engineering, National University of Singapore, Singapore
{eleaks,akdas,akash}@nus.edu.sg

ABSTRACT

Dynamic voltage and frequency scaling (DVFS) offers great potential for optimizing the energy efficiency of Multiprocessor Systems-on-Chip (MPSoCs). The conventional approaches for processor voltage and frequency adjustment are not suitable for streaming multimedia applications due to the cyclic nature of dependencies in the executing tasks which can potentially violate the throughput constraints. In this paper, we propose a methodology that applies DVFS for such cyclic dependent tasks. The methodology involves an off-line analysis that assumes worst-case execution times of tasks to identify the executions that can be slowed down and an on-line analysis to utilize the slacks arising from tasks that finish their execution before the worst-case execution times. Thus, the methodology minimizes energy consumption during both off-line and on-line analysis while satisfying the throughput constraints. Experiments based on models of real-life streaming multimedia applications show that the proposed methodology reduces the overall energy consumption by 43% when compared to existing approaches.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time systems and embedded systems

General Terms

Algorithms, Design, Management, Performance

Keywords

Multiprocessor Systems-on-Chip, streaming applications, energy consumption, throughput constraint

1. INTRODUCTION

Modern embedded systems (e.g., mobile phones, tablets) need to support a number of streaming multimedia applications. In order to satisfy the ever increasing performance (throughput) constraints of the applications, MPSoC based systems are being designed. Since such systems are usually operated by stand-alone power supply like battery, minimizing energy consumption during their design and operation is important in order to increase the operational time.

Several efforts have been made to minimize energy consumption of battery-operated MPSoCs. These efforts use system-level energy optimization techniques such as efficient scheduling and DVFS. Many advanced processors support DVFS with multiple voltage levels, which are used in both general-purpose and embedded computing domains [1–4,17]. For applying DVFS in MPSoCs, the voltage and frequency of one or more processors is adjusted depending upon the workload of processors while satisfying the throughput constraint [5]. It has been observed that lowering the voltage by half might lead to eight times reduction in power consumptions with the linear reduction in maximum operating frequency of a CMOS circuit [7]. The DVFS approaches have also been validated [29]. Executing at lower frequency results in stretched (slowed down) execution, which should not violate the timing constraints such as throughput.

Existing DVFS techniques apply off-line [18] [14] or on-line [8] [23] voltage and frequency scaling. Off-line DVFS techniques estimate the voltage scaling (VS) levels at compile time (design-time) with the knowledge of specific task timings, e.g. worst-case task execution times. On-line DVFS techniques make the voltage and frequency scaling decisions at run-time based on the slack arising from tasks finishing before the worst-case execution-times (WCETs). The off-line DVFS techniques cannot exploit the slacks arising at run-time and the on-line DVFS strategies utilize only limited information to keep a minimum run-time computation overhead. Further, most of the existing DVFS strategies are applicable only to applications described as independent tasks and task graphs represented as directed acyclic graphs (DAGs). Additionally, they don't take the actual VS overhead into account. Such strategies are not applicable for streaming applications that normally exhibit *cyclic* dependencies amongst the tasks and need to be executed in modern embedded systems such as smart phones and tablets. In [24] and [20], DVFS techniques that are applicable on the applications having cyclic dependencies amongst tasks are presented but the techniques have several limitations. For example, in [24], applications need to be described as *homogeneous* synchronous dataflow graphs (HSDFs) [19] that impose high computation complexity, tasks are bound by WCET and only static-slack (off-line slack) created by the difference between application's desired and obtained throughput is used for energy reduction. In [20], only on-line DVFS is applied while considering a uni-processor system.

Contribution: This paper addresses shortcomings of existing DVFS techniques and proposes a methodology that applies both off-line and on-line DVFS to reduce the energy consumption for applications containing cyclic dependent tasks. In the off-line analysis, the tasks are assumed to have WCETs and DVFS is applied in two phases. In the first phase, the executions of tasks are stretched (slowed down) by applying suitable voltage scaling (VS) without violating the throughput constraint. The second phase analyzes execution traces of application tasks & their dependencies to identify the parallel executions that can be slowed down without violating the throughput constraint and applies appropriate VS. The updated execution trace by applying the off-line analysis is used to apply on-line DVFS where VS is further applied based on the dynamically created slacks due to tasks finishing earlier than their WCETs. VS assumes linear frequency scaling and the methodology takes VS overhead captured from manufacturer's datasheet into account, which is explained in section 5. We evaluate the proposed methodology for streaming multimedia applications represented as SDFs. The methodology applies VS directly on SDFs without converting them into HSDFs in order to reduce the computation cost and it is general enough to be applied to applications described as independent tasks and DAGs as well.

The remainder of the paper is organized as follows. Section 2 reviews the literature in the direction of off-line and on-line DVFS. Section 3 introduces the preliminaries necessary to understand the work. Section 4 presents the proposed DVFS methodology. The experimental results to evaluate our methodology are presented in Section 5. Section 6 concludes the paper.

2. RELATED WORK

Amongst the earliest works to apply DVFS, Yao et. al. [32] proposed an algorithm to compute optimal static slow down schedule for a set of tasks. Several extensions were undertaken for periodic and aperiodic tasks [6] [26]. Inter-task DVFS [34] [33] and intra-task DVFS [27] [20] have been applied to execute a task at a single and multiple voltage levels, respectively. Recent work also incorporates leakage power aware DVFS [16] [9]. Most of the aforementioned DVFS strategies target a uni-processor system and apply off-line or on-line DVFS to reduce the energy consumption [31]. Simply extending them for MPSoCs leads to increased complexity and inefficiency.

A large body of research exists for applying off-line DVFS while targeting MPSoC [14, 18, 21, 24, 25]. These strategies assume fixed execution time (e.g. WCET) for each task and thus cannot be applied to reduce energy consumption at run-time where tasks have varying execution times.

Dynamic slack reclamation techniques to apply on-line DVFS while targeting MPSoC have also been proposed [8, 23, 35]. These techniques utilize dynamically created slack (due to earlier finish of the tasks) to reduce overall energy consumption. In order to maximize utilization of the slack created on a processor, it is shared amongst other processors or forwarded to later executing tasks. These on-line DVFS strategies don't use any off-line analysis and thus impose high run-time scheduling overhead and reduce energy consumption only due to dynamically created slacks.

On-line DVFS strategies using off-line analysis are presented for MPSoC [10, 11, 22]. The off-line analysis aims at minimizing expected energy consumption. In [10], an off-line analysis phase calculates expected future slack and computation of future tasks by taking average and worst case execution timings. In [22] and [11], the off-line analysis constructs static schedules to be used at run-time. The schedule is constructed by considering critical path and task execution orders. Since these strategies use off-line analysis, run-time scheduling overhead gets reduced and energy consumption is optimized during both design-time and run-time. However, they cannot be efficiently applied to applications containing cyclic dependent tasks as they are developed while targeting independent tasks or DAGs.

In contrast to above strategies, our approach applies both off-line and on-line DVFS for MPSoCs while targeting applications containing cyclic dependent tasks, which are modeled as Synchronous Dataflow Graphs (SDFGs) [19]. In [20] and [24], the applications are modeled as SDF but energy is reduced by applying either off-line [24] or on-line [20] DVFS. Further, the strategy in [20] considers uni-processor system and in [24] is applicable to homogeneous SDFs (HSDFs) that need to be derived from SDFs. Our approach is applicable to applications represented as independent tasks and DAGs as well.

3. PRELIMINARIES

This section provides a brief overview of the MPSoC platform & application model and challenges involved in applying DVFS.

3.1 MPSoC Platform & Application Model

The MPSoC platform is modeled as tile-based architecture [12]. Fig. 1 shows an example platform containing three tiles having DVFS capabilities. Each tile consists of a processor (P), a local memory (M, size in bits) and a network interface (NI). The DVFS controller adjusts voltage and frequency of processors to reduce the overall energy consumption similar to the one supported in Intel's XScale processor [2]. In order to facilitate communication amongst tiles, they are connected to an interconnection network through the NI. The interconnection network provides end-to-end connections between the tiles. However, the latencies of connections can be modeled according to different network-on-chips (NoCs).

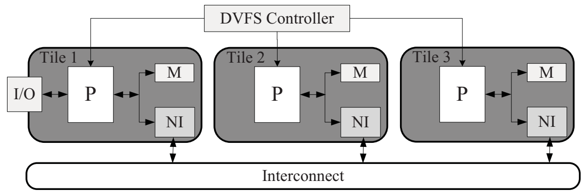

Figure 1: Example MPSoC platform.

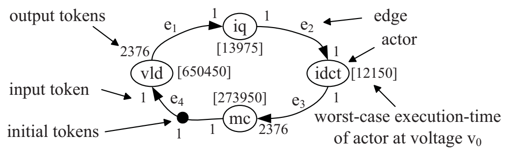

Figure 2: SDFG model of an H.263 decoder.

The streaming multimedia applications with timing constraints are modeled as SDFGs [19]. Fig. 2 shows SDFG model of H.263 decoder that contains cyclic dependent tasks. The nodes (*vld, iq, idct* & *mc*) and edges (e_1, e_2, e_3 & e_4) model tasks and dependencies respectively. The nodes are referred to as *actors* that communicate with *tokens* sent from one actor to another through the edges. Each actor has its attributes WCET and memory requirement when mapped on a tile operating at a particular voltage level. Each edge has following attributes: size of a token, memory needed on the tile when connected actors are allocated to the same tile, memory needed in source and destination tiles when connected actors are allocated to different tiles and respective bandwidth requirements between the tiles. An actor *fires* (executes) when there are sufficient input tokens on all of its input edges and sufficient buffer space on all of its output connections, and in turn the actor consumes a fixed amount of tokens from the input edges and produces a fixed amount of tokens on the output edges. These token amounts are referred to as *rates*. An edge may contain *initial tokens*.

Throughput of an application is determined as the inverse of the long term period that is calculated as the average time needed for one iteration of the application. An iteration is defined as the minimum non-zero execution such that the original state of the SDFG is obtained. Period for the example H.263 decoder is equal to the summation of ExecTime(*vld*), 2376×ExecTime(*iq*), 2376×ExecTime(*idct*) and ExecTime(*mc*), where ExecTime is the WCET of respective actors. This period does not include network and memory access delays. It should be noted that actors *iq* and *idct* have to execute 2376 times in one iteration and the number of executions is referred to as *repetition vector* of the actor. An SDFG with a throughput of 1000 Hz has a period of 1 millisecond (ms), i.e. takes 1 ms to complete one iteration.

3.2 DVFS for Applications modeled as Cyclic SDF Graphs

The DVFS process is applied for a given application to MPSoC platform mapping. In a mapping, actors are bound to tiles and edges to memory inside tiles or to connections in the platform. Applying voltage scaling (VS) on one or more tiles results in stretched (slowed down) execution of the bound actors. Modern processors support multiple voltage levels and thus several VS options exist for the same given mapping. Evaluating the throughput obtained with different VS options on each processor is time consuming. This imposes a challenge to rapidly identify the VS options that will lead to throughput satisfying scaling. Further, for a given application and MPSoC platform, as there are several possible mappings (actors to tiles allocations), the complete evaluation with different VS options on each processor might not be feasible within an acceptable time. In order to reduce

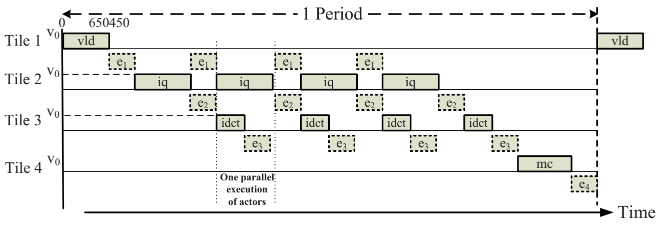

Figure 3: Execution trace of actors/edges of H.263 decoder.

the evaluation time, efficient mappings (providing optimal throughput) need to be identified and only appropriate VS options need to be applied.

In case of applications containing cyclic dependent actors, applying VS on a tile is very challenging as one needs to capture the VS effect on the execution of actors mapped on other tiles by taking the cyclic dependencies into account. Most of the existing DVFS strategies do not take such cyclic dependencies into account and thus cannot be applied to streaming multimedia applications such as H.263 decoder (Fig. 2). Fig. 3 shows the execution trace of the H.263 decoder mapped on a 4-tile MPSoC such that each actor is mapped on a separate processor tile operating at voltage v_0. The connections containing edges also operate at the same voltage. Each rectangle represents an execution such that its height corresponds to the operating voltage (pertaining to a frequency) and its length corresponds to the execution time. First, actor *vld* fires (executes) as it has sufficient input tokens on its incoming edge e_4. Thereafter, it generates 2376 tokens to be transferred through e_1 to process them one by one by *iq*. The transfer of tokens through edges and their processing by different actors follows the shown trace. For easier understanding, the shown trace considers rates as 4 in places of 2376 and thus actors *vld, iq, idct* & *mc* fire 1, 4, 4 & 1 times respectively during one period. Actor *vld* fires again after finishing the execution of actor *mc* and similar execution patterns are followed in the upcoming periods due to cyclic dependencies. The existing DVFS strategies cannot be applied on such executions.

4. PROPOSED DVFS METHODOLOGY

This section describes our DVFS methodology. In contrast to conventional existing DVFS methodologies, our methodology differs in following aspects: *1)* applies DVFS for applications containing cyclic dependent actors, *2)* applies DVFS by analyzing the execution traces of actors/edges, *3)* for a given application and MPSoC platform, applies off-line DVFS only for the optimal mappings in order to perform faster evaluation, *4)* off-line DVFS results are used to apply on-line DVFS for the dynamically created slacks, and *5)* considers actual VS overhead.

An overview of our DVFS flow is presented in Fig. 4. The DVFS flow optimizes energy consumption for each application to be supported on a MPSoC platform. The applications are evaluated one after another by using the same DVFS flow. The flow takes an application (*Application Model*) & a platform (*Platform Model*) as input and performs design-time and run-time energy optimization for different mappings. At design-time, the flow first evaluates mappings for their throughput and energy consumption. The mappings are generated by using the strategy proposed in [28] as it discards evaluation of inefficient mappings (providing less throughput) and performs faster evaluation without missing the efficient mappings. Then, off-line analysis is applied for the Pareto-optimal mappings that represent resource-throughput trade-offs. At run-time, first, the best mapping (having maximum throughput) is selected depending upon the throughput constraint and available platform resources to configure the platform. Then, on-line DVFS is applied to utilize the dynamically created slacks. Throughput and energy consumption computation for each mapping are done as follows.

Throughput Computation: The throughput for a mapping is computed by taking the resource allocations of actors/edges on the platform into account. For each platform tile, first, static-order schedule that orders the execution of bound actors is constructed. Then, all the binding and scheduling decisions are modeled in a graph called binding-aware SDFG. Finally, throughput is computed by self-timed state-space exploration of the binding-aware SDFG [13]. During the self-timed execution, states visited are examined and stored until a recurrent state is found. The throughput is computed from the periodic part of the state-space.

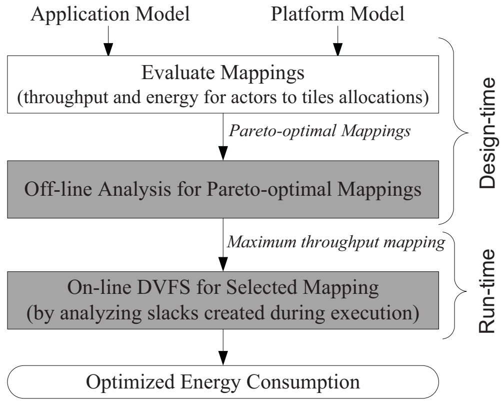
Application Model
Platform Model
Evaluate Mappings
(throughput and energy for actors to tiles allocations)
Pareto-optimal Mappings
Off-line Analysis for Pareto-optimal Mappings
Design-time
Maximum throughput mapping
On-line DVFS for Selected Mapping
(by analyzing slacks created during execution)
Run-time
Optimized Energy Consumption

Figure 4: Proposed DVFS flow.

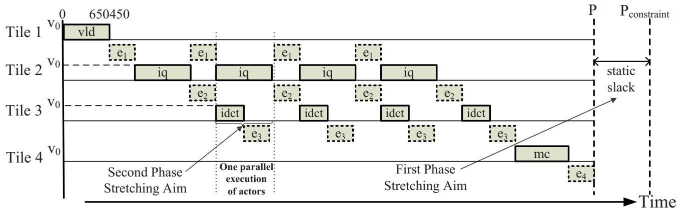

Figure 5: Aim of first and second phase off-line analysis.

Energy Consumption Computation: The total energy consumption for a mapping is computed as the sum of communication and computation energy for one iteration (periodic execution) of the application. *Communication energy* is required to transfer data (tokens) from source tile to destination tile and the energy required to process the transferred token on the destination tile is referred to as the *computation energy.* The detailed approach to compute total energy consumption is provided in Appendix A.

4.1 Off-line Analysis

The off-line analysis strategy applies voltage scaling (VS) for each Pareto-optimal mapping in two different phases. The first phase aims to utilize the static slack while maximizing the energy savings as shown in Fig. 5. The static slack is defined as the difference between period ($P_{constraint}$) corresponding to throughput constraint and period (P) corresponding to obtained throughput, where period is equal to 1/throughput. The timing constraint $P_{constraint}$ has also been referred to as deadline constraint. The second phase aims to stretch the slower parallel executions.

4.1.1 First Phase Analysis

In the first phase, the lowest possible VS levels for different tiles are identified while satisfying the throughput constraint and maximizing the energy savings. Towards this, we have proposed a Greedy and an Integer Linear Programming (ILP) formulation based approach.

Greedy Approach

The greedy approach is presented in Algorithm 1. The algorithm takes throughput constraint (τ), VS levels (V) and WCETs of actors at different VS levels (ET) as input and identifies the VS levels to be applied. For the given Pareto-optimal mapping, first, the tiles containing actor(s) are selected. Then, for each selected tile, different available VS

ALGORITHM 1: Greedy Analysis

```
Input: throughput constraint $\tau$, $V = \{v_i | \forall i \in [1, \cdots, n]\}$,
        $ET = \{WCET[a] \rightarrow t_{v_i | \forall i} | \forall actors\}$.
Output: VS levels of tiles.
Select tiles containing actor(s);
repeat
    for each selected tile $t$ whose VS level is not fixed do
        for each VS level $v_i$ do
            Apply VS $v_i$ on $t$ and compute throughput $thrMap$;
            if $thrMap > \tau$ then
                Calculate energy savings $ES[t][vi]$ from equation 1;
            end
        end
    end
    Find tile $t_f$ & VS level $v_f$ corresponding to maximum ES[][];
    Fix voltage of $t_f$ to $v_f$;
until VS levels of all selected tiles are not fixed;
```

levels are applied and throughput of the mapping ($thrMap$) is computed. If an applied VS on a tile satisfies the throughput constraint then energy savings for all the actors mapped on the tile (a_0,...,a_m) during one periodic execution is calculated from equation 1, where v_0 and v_i represents the initial and applied voltages respectively. Thereafter, the tile and its VS level corresponding to maximum energy savings is found. The same process is repeated to find VS levels of other tiles.

$$ES[t][v_i] = \sum_{a=a_0}^{a_m} repV[a] \times [(ET[a] \rightarrow t_{v_0}) \times (pow \rightarrow t_{v_0}) - (ET[a] \rightarrow t_{v_i}) \times (pow \rightarrow t_{v_i})] \quad (1)$$

The **complexity** of the Greedy analysis in terms of number of used (selected) tiles n in the mapping and available VS levels l has been evaluated. For a given value of n and l, the worst-case complexity (C) is calculated as the *maximum number of throughput computations* by equation 2. The complexity of the analysis is O(ln^2).

$$\begin{aligned} C &= n \times l + (n-1) \times l + ... + 2 \times l + 1 \times l \\ &= l \times \sum_{p=1}^{n} p = l\left(\frac{n^2}{2} + \frac{n}{2}\right) \end{aligned} \quad (2)$$

ILP Formulation
To facilitate the ILP formulation, two tables – *slack use table* and *energy savings table* are defined (see Appendix B). There are n rows and l columns for both the table, where n and l are the number of used tiles and supported voltage levels respectively. Each entry (i, j) of the slack used table (SU) is computed as follows.

$$\begin{aligned} SU(i,j) &= computePeriod(t_i \rightarrow v_j | t_k \rightarrow v_0) - P \\ &\quad \forall k \in [1 \cdots n], k \neq i \\ P &= computePeriod(t_i \rightarrow v_0, \forall i \in [1 \cdots n]) \end{aligned} \quad (3)$$

where $computePeriod$ function computes period as 1/throughput with tile t_i assigned voltage level v_j and all other tiles set to voltage v_0 (the highest supported voltage level in the platform). The slack used is obtained by subtracting this period with the period obtained by setting all tile voltages to v_0. The energy savings table (ES) is computed in a similar way as shown below.

$$\begin{aligned} ES(i,j) &= E - computeEnergy(t_i \rightarrow v_j | t_k \rightarrow v_0) \\ &\quad \forall k \in [1 \cdots n], k \neq i \\ E &= computeEnergy(t_i \rightarrow v_0, \forall i \in [1 \cdots n]) \end{aligned} \quad (4)$$

Binary Variables: $X_{ij},\ i \in [1, n],\ j \in [1, l]$
Objective: Maximize z = $\sum_{ij} X_{ij} \times ES(i, j)$
Constraints:
One element from each row and column of the table are to be selected.

$$\sum_{i=1}^{n} X_{ij} = 1; \quad \sum_{j=1}^{l} X_{ij} = 1 \quad (5)$$

Total slack distributed on the tiles must be less than or equal to the available slack.

$$\sum_{ij} X_{ij} \times SU(i,j) \leq D - P \quad (6)$$

where D is deadline ($P_{constraint}$) of the given application.

The *number of throughput (period) computations* in the ILP approach is $n{\times}l$, whereas for the Greedy approach it is O(ln^2). The ILP approach uses $n{\times}l$ throughput computations to fill the slack used table.

4.1.2 Second Phase Analysis

In the second phase, the executions that can be further slowed down without violating the deadline ($P_{constraint}$) are identified to apply the appropriate VS while maximizing the utilization of the remained static slack. The remaining slack is defined as difference between $P_{constraint}$ and period obtained after applying first phase analysis. The second phase of the off-line DVFS technique is presented in Algorithm 2. The algorithm takes captured execution traces of actors/edges operating at different voltage levels after applying first phase analysis, WCETs of actors at available VS levels & static slack as input and provides updated execution traces after applying additional VS. The execution traces for each actor and edge is captured as the start time, end time and operating voltage of the active executions (firings) during one periodic execution. The algorithm identifies the executions that can be slowed down and applies VS on them. The smaller parallel executions are the ones that can be stretched without significantly stretching the overall execution trace. In Fig. 3, parallel executions of *idct* (with *iq*) can be stretched till the executions of *iq* by applying a suitable voltage scaling if the deadline does not get violated.

The algorithm first finds actors executing in parallel by analyzing the execution traces. Then, the actors are sorted in ascending order of their WCETs in order to apply voltage scaling from the actor having lowest WCET to the actors having higher WCET until all the parallel executing actors are covered. The execution trace of the actor having lowest WCET is stretched by applying an appropriate voltage scaling provided the dependency of the actor is neither longer than connected outgoing edge execution (i.e., just after the outgoing edge firing, there should not be any dependent firing) in some of the actor firings, nor on other actors mapped on the same tile. The stretching (slow down) of all the firings is done based on the execution of the parallel executing actor having next higher WCET and the static slack. Stretching slower parallel executions does not affect the overall throughput and thus the same has been considered. Then, the algorithm selects the actor with next higher WCET and applies voltage scaling on all the earlier selected actors in the same manner. The same process is repeated till the parallel executing actor having highest WCET is not selected as the actor having next higher WCET. Thus, the algorithm applies recurrent voltage scaling on most of the parallel executing actors, resulting in maximum energy consumption reduction.

The demonstration of the off-line analysis for the example H.263 decoder (Fig. 2) is presented in Appendix C. The analysis applies same voltage scaling for all firings of an actor on a tile in order to avoid the overhead of keeping track of the actor firings with different WCETs and VS levels to be used at run-time. This consideration reduces the voltage switching overhead as well.

4.2 On-line DVFS

The off-line processed traces with voltage level information are used to apply on-line DVFS based on dynamically created slacks. The on-line technique applies further voltage scaling to utilize dynamically created slacks due to finishing of the actors earlier to their WCETs at run-time. The technique is presented in Algorithm 3. For a given application

ALGORITHM 2: Trace-based Analysis

```
Input: Execution traces operating at different voltages,
       $ET = \{WCET[a] \rightarrow t_{v_i|\forall i} | \forall actors\}$ and static slack.
Output: Updated execution traces with new operating voltages.
Find actors executing in parallel and their parallel executions;
Sort the actors based on their WCETs at the operating voltages;
Select actor $a$ having lowest WCET to stretch its execution;
repeat
    Select actor with next higher WCET as current actor $a_c$;
    for each earlier selected actor $a_i$ except $a_c$ do
        if firings of other actors are not dependent on firing of
        outgoing edge of $a_i$ AND firing of $a_i$ is not dependent on
        other actor on the same tile then
            Determine stretching of $a_i$ firings by considering its
            parallel firings with $a_c$ and static slack;
            Apply appropriate VS for $a_i$ firings based on the
            determined stretching;
        end
    end
until all parallel executing actors not covered;
```

ALGORITHM 3: On-line DVFS

```
for each firing on tile $t$ do
    Calculate $AET = finish\_time - start\_time$;
    Compute $slack$ += $(WCET - AET)$;
    if $slack > WCVST$ then
        Compute reduced speed $f'$ for the next firing by Equation 7;
        Apply VS level $v_i$ for next firing on $t$ such that (speed at
        $v_i) \geq f'$;
        $slack = 0$;
    end
end
```

to be supported onto a platform at run-time, the technique applies voltage scaling by using off-line processed execution traces for the best (maximum) throughput mapping that is selected from the Pareto-optimal mappings based on the available platform tiles and throughput constraint. The execution traces contain information of actors' & edges' firings by considering WCETs and their operating voltages on the tiles. At run-time, these operating voltages are used for the first firing on the different tiles. Each tile invokes the algorithm whenever the tile finishes a firing of the mapped actor(s).

The implementation of on-line DVFS considers a Ready Queue (RQ) that contains all ready firings of actors, an array to keep initial speed of tile for the firings, and an array to store start time of the firings, for each tile. For each tile, all the firings are put into RQ in the order of their execution priorities. When a firing is finished on a tile, a slack (worst-case execution time (WCET) - actual execution time (ACT)) gets created and the tile starts with the next firing from the RQ by following Algorithm 3. If the created slack is greater than the worst-case voltage switching time (WCVST), then the tile calculates its speed (f') to execute the next firing by Equation 7, where f' and f represent the reduced and earlier speed respectively. Otherwise, the created slack is accumulated to be used for further firings (Algorithm 3). Such consideration avoids non-beneficial DVFS. Calculation of f' considers WCVST in order to take voltage switching overhead into account. An appropriate voltage scaling is applied for the next firing based on f' so as to avoid violation of the deadline. As the voltage and frequency levels are discrete, a scaling level having frequency higher and closer to f' is applied.

$$f' = f \times \frac{WCET}{WCET + slack + WCVST} \qquad (7)$$

The runtime overhead of the algorithm depends upon the number of used tiles (m) by the application and the number of firings (n) on the different tiles. DVFS is not applied on the first firing on each tile as it starts with the off-line suggested voltage/speed. Therefore, the algorithm has worst-case runtime overhead for a maximum of (n-m) firings, where total time for each firing consists of time taken to calculate AET, slack, reduced speed f', and WCVST.

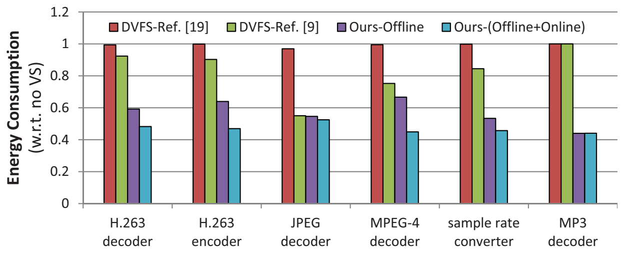

Figure 6: Energy consumption for different applications.

The algorithm does not use slack on one tile for another tile to avoid the overhead of managing its effect on the dependent execution on different tiles. Thus, the algorithm applies efficient voltage scaling leading to reduced energy consumption. The on-line DVFS for different firings of actor *idct* of H.263 decoder is demonstrated in Appendix D.

5. PERFORMANCE EVALUATION

The proposed DVFS methodology has been implemented as an extension of the publicly available SDF^3 tool set [30]. As a benchmark to evaluate the quality of the methodology, models of real-life streaming multimedia applications H.263 decoder (4 actors), H.263 encoder (5 actors), MPEG-4 decoder (5 actors), JPEG decoder (6 actors), sample rate converter (6 actors) and MP3 decoder (14 actors) have been considered. For each application, its throughput requirement and WCETs of actors are known a priory and specified in the application model. The methodology considers DVFS-capable processors in the platform. Each platform tile contains StrongARM processor [1], which supports four voltage-frequency levels: 1.5V–206MHz, 1.4V–192MHz, 1.2V–162MHz and 1.1V–133MHz. The worst-case voltage switching time (WCVST) for each processor is considered as the voltage switching time of StrongARM processor [1]. The energy consumed during the voltage transition is considered negligible in comparison to the overall energy consumption. The AET of an actor firing is expressed as fraction (α) of it's WCET. For different actor firings, we vary α randomly from 0.6 to 1.0.

We present results obtained from our DVFS methodology and compare them with existing methodologies proposed in [20] and [10] in terms of total energy consumption. In order to realize energy savings by applying different DVFS methodologies, energy consumption has been estimated without applying voltage scaling (VS) as well. The strategy in [20] considers applications modeled as SDFGs but is applicable to a uni-processor system. It has been extended to multiprocessor system for a fair comparison. In [10], applications modeled as DAGs are considered. Based on the execution behavior of DAGs, the application models are translated into corresponding SDFGs in order to provide a fair comparison.

Fig. 6 shows energy consumption by different DVFS methodologies with respect to (w.r.t) energy consumption without applying VS. Our approach utilizes the Greedy algorithm for the first phase offline optimization. For each application, energy consumption is estimated for the mapping using the same number of tiles as the number of actors in the application. It can be observed that our methodology shows higher reduction in energy consumption for all the applications. Reduction in energy consumption by the methodologies in [20] and [10] is not significant as they have high penalty to consider the voltage switching time. On an average, our methodology reduces energy consumption by 52% and 43% when compared to methodologies in [20] and [10], respectively.

Next, we analyzed the effect of number of used tiles by the applications on the reduction in energy consumption. Fig. 7 shows energy consumption for the best (maximum throughput) mappings using different number of tiles for

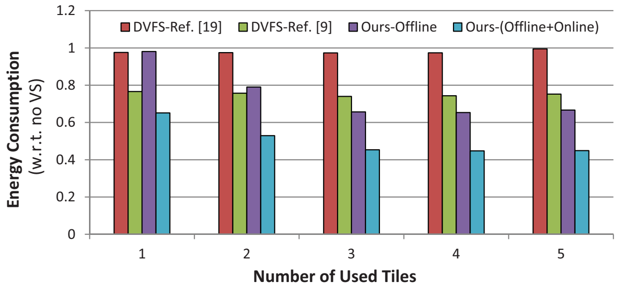

Figure 7: Energy consumption for different number of used titles.

MPEG-4 decoder when different DVFS methodologies are employed. The energy consumptions are shown with respect to no VS energy consumption. The shown results by our approach utilize ILP formulation for the first phase offline optimization. It can be observed that total energy savings (reduction) by our approach increases with the number of used tiles as the combinations of VS options on different tiles get increased, which facilitates for higher energy savings.

Effect of α has been analyzed on the energy savings. The evaluated energy consumption by different methodologies with varying α is presented in Appendix E. The runtime overhead of the on-line DVFS algorithm has been computed. For the H.263 decoder mapped on four tiles, the runtime overhead is 0.8 milliseconds when VS gets applied on 50 firings out of 190 total firings. Experiments have also been performed to analyze the effect of number of voltage levels on the energy savings by the Greedy and ILP approach proposed for the first phase off-line analysis. The number of voltage levels of strongARM [1] are restricted by taking 5 to 15 equally spaced voltage levels between its minimum (1.1V) and maximum (1.5V) operating voltage. Table 1 shows the effect of number of voltage levels on the energy savings for different applications. A couple of observations can be made from Table 1. First, at lower number of supported voltage levels, Greedy approach provides more energy savings as compared to the ILP approach for few applications (e.g. MPEG-4 decoder at 5 voltage levels). Second, ILP approach provides more energy savings than Greedy at higher number of supported voltage levels (e.g. 10 & 15) for all the applications. With large number of supported voltage levels, the Greedy approach finds a local maxima by fixing the voltage levels of a few tiles at the minimum and higher for other tiles, whereas ILP approach performs better (more uniform) distribution of voltage levels on the tiles providing higher energy savings. It has been observed that Greedy provides more energy savings than ILP for most of the applications when number of supported voltage levels is less than 5. For such case (VS levels < 5), trace-based analysis has been applied after the ILP approach and it has been observed that energy savings become closer to that of the Greedy approach.

6. CONCLUSION

We present a novel DVFS methodology for streaming applications that contain actors having cyclic dependencies. Moreover, the methodology is applicable to acyclic task graphs as well. We show that the methodology applies voltage scaling both at design-time and run-time while satisfying the temporal deadlines, and thus provides significant energy savings when compared to existing DVFS methodologies. In future, we plan to apply voltage scaling on execution of application edges while considering an interconnect supporting multiple voltage levels. We also plan to consider heterogeneous platform to accelerate some executions towards creating longer slacks, which can be used to apply further voltage scalings. These considerations will facilitate for further energy savings.

7. ACKNOWLEDGMENTS

This work is supported by Singapore Ministry of Education Academic Research Fund Tier 1 under grant number R-263-000-655-133.

Table 1: Energy savings (nJ) at different number of supported voltage levels

	5 voltage levels		10 voltage levels		15 voltage levels	
	ILP	Greedy	ILP	Greedy	ILP	Greedy
H.263 decoder	640	518	53135	474	52628	486
H.263 encoder	644	644	14153	889	5369	880
MPEG-4 decoder	1431	1723	87471	1561	86626	1546
JPEG decoder	36	36	1022	681	1448	337

8. REFERENCES

[1] Marvell, StrongARM 1100 processor, 1997. http://www.marvell.com/.

[2] Intel XScale 80200 Processor, 2001. http://www.intel.com/.

[3] Transmeta, Transmeta Crusoe Processor, 2001. http://www.transmeta.com/.

[4] ARM1176 Processor, 2004. http://www.arm.com/.

[5] A. Alimonda et al. A feedback-based approach to dvfs in data-flow applications. *IEEE TCAD*, pages 1691–1704, 2009.

[6] H. Aydin et al. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Comput.*, 53(5):584 – 600, may 2004.

[7] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *IEEE HICSS*, pages 288–297, 1995.

[8] J.-J. Chen et al. Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors. In *IEEE SUTC*, pages 358–367, 2006.

[9] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *IEEE ICCAD*, pages 289–294, 2007.

[10] P. Choudhury et al. Online dynamic voltage scaling using task graph mapping analysis for multiprocessors. In *IEEE VLSID*, pages 89–94, 2007.

[11] J. Cong and K. Gururaj. Energy efficient multiprocessor task scheduling under input-dependent variation. In *DATE*, pages 411 –416, 2009.

[12] D. Culler et al. *Parallel computer architecture: a hardware/software approach.* Morgan Kaufmann Pub, 1999.

[13] A. H. Ghamarian et al. Throughput Analysis of Synchronous Data Flow Graphs. In *IEEE ACSD*, pages 25–36, 2006.

[14] F. Gruian. System-level design methods for low-energy architectures containing variable voltage processors. In *Springer PACS*, pages 1–12, 2001.

[15] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE*, pages 234–239, 2004.

[16] R. Jejurikar and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *DAC*, pages 111–116, 2005.

[17] J. A. Kahle et al. Introduction to the cell multiprocessor. *IBM J. Res. Dev.*, 49:589–604, 2005.

[18] P. Langen and B. Juurlink. Leakage-aware multiprocessor scheduling. *J. Signal Process. Syst.*, 57:73–88, 2009.

[19] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36:24–35, 1987.

[20] S. Lee et al. An intra-task dynamic voltage scaling method for soc design with hierarchical fsm and synchronous dataflow model. In *ISLPED*, pages 84 – 87, 2002.

[21] K. Li. Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed. *IEEE TPDS*, 19(11):1484 –1497, nov. 2008.

[22] J. Luo and N. K. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *IEEE ASP-DAC*, pages 719–726, 2002.

[23] P. Malani et al. Adaptive scheduling and voltage scaling for multiprocessor real-time applications with non-deterministic workload. In *DATE*, pages 652–657, 2008.

[24] A. Nelson et al. Power minimisation for real-time dataflow applications. In *IEEE DSD*, pages 117–124, 2011.

[25] D. Shin and J. Kim. Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In *ISLPED*, pages 408–413, 2003.

[26] D. Shin and J. Kim. Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. In *IEEE ASP-DAC*, pages 653–658, 2004.

[27] D. Shin and J. Kim. Optimizing intratask voltage scheduling using profile and data-flow information. *IEEE TCAD*, 26(2):369 –385, feb. 2007.

[28] A. K. Singh et al. A Hybrid Strategy for Mapping Multiple Throughput-constrained Applications on MPSoCs. In *ACM CASES*, pages 175–184, 2011.

[29] D. C. Snowdon et al. Power management and dynamic voltage scaling: Myths and facts. In *PARC*, New Jersey, USA, 2005.

[30] S. Stuijk et al. SDF^3: SDF For Free. In *IEEE ACSD*, pages 276–278, 2006.

[31] W. Wang et al. Energy-aware dynamic slack allocation for real-time multitasking systems. *Sust. Comp.: Infor. and Sys.*, 2:128 – 137, 2012.

[32] F. Yao et al. A scheduling model for reduced cpu energy. In *IEEE FOCS*, pages 374–, 1995.

[33] S. Zhang et al. Approximation algorithms for power minimization of earliest deadline first and rate monotonic schedules. In *ISLPED*, pages 225–230, 2007.

[34] X. Zhong and C.-Z. Xu. System-wide energy minimization for real-time tasks: Lower bound and approximation. *ACM Trans. Embed. Comput. Syst.*, 7:28:1–28:24, 2008.

[35] D. Zhu et al. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE TPDS*, 14:686–700, 2003.

APPENDIX

A. TOTAL ENERGY CONSUMPTION

Communication energy is required to transfer tokens from one tile to another through the connections. In between two tiles, the communication has to take place when actors mapped on them need to communicate with each other. The communication energy for each edge e mapped to a connection c is estimated as product of the number of tokens (in bits) to be transferred through c, delay (D) and power consumption (P_{bit}) of c for transferring one bit while operating at voltage v. Total communication energy for all the edges is estimated from equation 8. The number of transferred tokens for an edge is computed as the product of repetition vector ($repV$) of source (or destination) actor and source (or destination) port rate (equation 9). The power required to transfer one bit is estimated from [15].

$$E_{comm} = \sum[\{nrTokens[e] \times tokenSize[e]\} \times (D \rightarrow c_v) \times (P_{bit} \rightarrow c_v)] \tag{8}$$

$$nrTokens[e] = repV[e \rightarrow srcActor] \times (e \rightarrow srcPortRate) \tag{9}$$

Computation energy is required to process the transferred token on the destination tile after it is received and able to fire (execute) the mapped actor. Computation energy for each actor a mapped to tile t is estimated as product of the number of executions of a ($repV[a]$), execution time ($ET[a]$) and power consumption (pow) on tile t operating at voltage v. Total computation energy for all actors is estimated from equation 10. Power consumption on tile is estimated by equation 11, where C, v and f denote average load capacitance, supply voltage and operating frequency, respectively.

$$E_{comp} = \sum[repV[a] \times (ET[a] \rightarrow t_v) \times (pow \rightarrow t_v)] \tag{10}$$

$$pow \rightarrow t_v = C \times v^2 \times f \tag{11}$$

Total energy consumption is measured as sum of communication and computation energy. The total energy consumption does not include static energy. In our approach, we focus on mapping of applications on the architecture after it is designed. So we cannot optimize static energy consumption and focus on optimizing only dynamic energy consumption.

B. ILP FORMULATION USED TABLES

The two tables **slack use table** and **energy savings table** as defined in Table 2 and Table 3 respectively are used to formulate the ILP.

Table 2: Slack used table

Tiles	Slack used in *ms*			
	v_0	v_1	$\cdots$	v_{l-1}
t_0	0	140	...	200
t_1	0	35	...	150
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
t_n	0	100	...	140

Table 3: Energy savings table

Tiles	Energy savings in *nJ*			
	v_0	v_1	$\cdots$	v_{l-1}
t_0	0	11,000	...	40,0000
t_1	0	135,000	...	150,000
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
t_n	0	15,0000	...	50,000

C. OFF-LINE ANALYSIS DEMONSTRATION

The demonstration considers one actor to one tile mapping (Fig. 3). Fig. 8 shows the execution traces after applying the off-line analysis (first & second phase). The first phase identifies the VS levels to be applied on different tiles while satisfying the deadline. The identified VS levels for each tile are assumed as v_1. The second phase analyzes the execution traces operating at voltage v_1 and provides the updated traces satisfying the deadline. The analysis (Algorithm 2) finds parallel executing actors as *iq* & *idct*, and applies voltage scaling v_2 ($< v_1$) for actor *idct* as it has lower WCET. Executions of *idct* are stretched till the execution of *iq* since the outgoing edge of *idct* does not affect most of other executions.

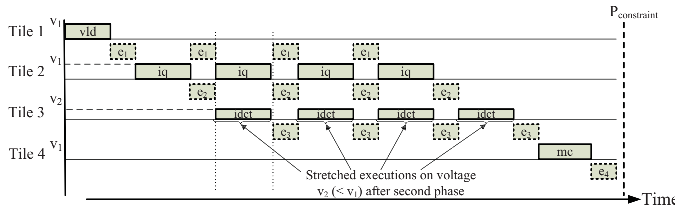

Figure 8: Execution trace of actors/edges of H.263 decoder after applying off-line DVFS.

D. ON-LINE ANALYSIS DEMONSTRATION

Fig. 9 shows the demonstration of on-line DVFS for different firings of actor *idct* of H.263 decoder mapped on a tile in order to apply further voltage scaling. It can be seen that if a firing finishes earlier than its WCET then created slack is used to apply DVFS on the next firing. The voltage scaling for the next firing is decided based on the reduced speed for the firing calculated by Equation 7. The voltage switching overhead has also been considered for the next firing. It can also be observed that overall time for all firings get reduced, i.e. execution trace gets shrunk. Thus, better throughput is obtained in addition to reduced energy consumption after applying on-line DVFS.

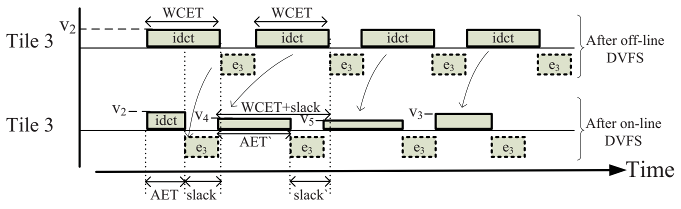

Figure 9: Applying on-line DVFS for different firings of an actor.

E. EVALUATION WITH DIFFERENT α

Effect of α has been analyzed on the energy savings. Experiments have been performed for different fixed values of α ranging from 0.1 to 1.0 at an interval of 0.1. At a fixed α, all actor firings assume the same value of α. Fig. 10 shows energy consumption by different methodologies for the maximum throughput mapping of MPEG-4 decoder that uses 3 tiles. A couple of observations can be made from Fig. 10. First, energy savings (reduction) is higher at lower values of α and decreases with higher values of α. Second, at α equal to 1, the methodologies in [20] and [10] do not provide energy savings as no slack is created at run-time, which contributes to the energy savings. However, our methodology provides energy savings by the off-line analysis.

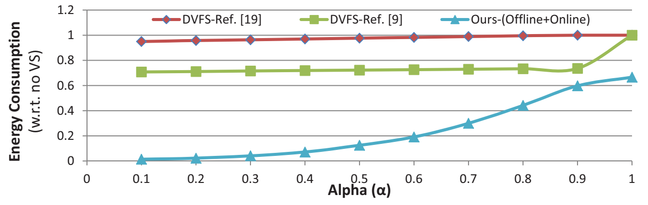

Figure 10: Energy consumption at different α.